Low Cost Built In Self Test for Public Key Crypto Cores

Duško Karaklajić, Miroslav Knežević and Ingrid Verbauwhede
Katholieke Universiteit Leuven, ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
Email: firstname.lastname@esat.kuleuven.be

***Abstract*—The testability of the cryptographic cores brings in an extra dimension to the process of digital circuits testing – security. The benefits of the classical methods such as the scan-chain method introduce new vulnerabilities concerning the data protection. The Built-In Self-Test (BIST) is considered to be the most suitable countermeasure for this purpose.**

In this work we propose the use of a digit-serial multiplier over GF(2^m), that is at the heart of many public-key cryptosystems, as a basic building block for the BIST circuitry. We show how the multiplier can be configured to operate as a Test Pattern Generator and a Signature Analyzer. Furthermore, the multiplier becomes a fully self-testable design. All the additional features come at the cost of only a few extra gates. With the hardware overhead of 0.33 % this approach makes the multiplier perfectly suitable for the low-end embedded devices.

***Keywords*-Built-In Self-Test; Pseudorandom Testing; Security; Public-Key Cryptography.**

I. Introduction

As circuits become larger and more complex, the process of testing a digital system remains a consistently growing problem. Design for testability (DFT) aims at providing an efficient testing method with a minimal cost. Testability is considered during design and needs to be addressed with the design of the rest of the system. The most common method for delivering the test data from the circuit inputs is called a scan-design. The data is delivered through registers that are connected to one or more scan-chains. The method seems to be very efficient, introducing a small hardware overhead and is especially suitable for embedded devices.

At the same time, as the embedded systems evolve from isolated devices to always-on networked devices, security becomes a paramount issue. Testing the cryptographic applications brings in an extra dimension – security. The most popular DFT techniques today, such as scan-chains, provide an observability of the internal nodes from the I/O pins of a device. As the embedded devices often store and process confidential data, such an approach is inadequate. There are several scan-chain based attacks published in the literature [19], [2].

The Built-In Self-Test (BIST) is considered to be the most suitable approach for testing the cryptographic cores. Since the internal registers of the device can not be accessed from the BIST I/O interface, it provides an inherent protection from the scan-chain based attacks. There are generally three approaches of the BIST testing: Exhaustive, Deterministic and Pseudorandom. The exhaustive approach achieves 100 % fault coverage but is exponential in number of the circuit inputs and becomes impractical for circuits with large number of I/O pins. The deterministic approach analyzes a Circuit Under Test (CUT) prior to testing and determines the appropriate test set to be applied. Finally, the pseudorandom approach is based on applying pseudorandom patterns as the test stimuli. A design is said to be pseudorandom pattern testable if the application of a fixed length sequence of pseudorandom patterns will detect, with a high confidence level, all faults of a particular fault class [15]. Due to the simplicity and the efficiency of its implementation this approach is very efficient testing solution.

A class of circuits which exposes excellent pseudorandom testability properties are cryptographic cores [16]. Specific structures and operations used for the implementation of cryptographic algorithms enable good random patterns propagation thus providing high testability. So far, only cores supporting the symmetric-key cryptography (*e.g.* block ciphers) were examined for the BIST purpose [4], [11], [16]. To the best of our knowledge, this work is the first attempt to use a public-key cryptographic core for a testing purpose. As a modular multiplication is at the heart of many public-key algorithms such as RSA [13], Elliptic Curve Cryptography (ECC) [8], [10], Diffie-Hellman, ElGamal, Schnorr, DSA [9], we use a digit-serial multiplier as a starting point and build the whole BIST circuitry based on it. We specifically target the low-end embedded devices and hence consider the ECC to be an appropriate choice for the public-key cryptosystem. Therefore, we start with a digit-serial multiplier over GF(2^m) that is at the core of the ECC algorithm.

II. Preliminaries

In order to make the following discussion easier, we first introduce a basic concept of the BIST and discuss the stuck-at fault model. Second, we explain basics of the modular arithmetic over GF(2^m) and introduce some notations that are used further throughout the paper. Finally, we provide some definitions and mention the related work of the random pattern testability which we later exploit in our model.

A. Built-In Self-Test and the Stuck-At Fault Model

Figure 1 represents the basic concept of the Built-In Self-Test environment. A BIST circuitry consists of a Test Pattern Generator (TPG) and a Signature Analyzer (SA). The TPG is used to produce test patterns which are fed into the Circuit Under Test (CUT). The SA compacts the CUT's

responses into a single signature, which is then compared to the precomputed golden value. Besides the basic components the BIST infrastructure also comprises the controller for scheduling different testing phases and the comparator for the final comparison. As the controller and comparator are common for almost all testing environments, we focus on the basic BIST components, namely TPG and SA.

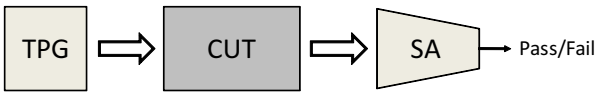

Figure 1. General BIST settings.

Concerning the choice of an appropriate fault model, Patel [12] points out that if a defect internal to a CMOS simple gate is detectable by some Boolean test then it is sufficient to have only the test vectors that test the stuck-at faults on the gate terminals. Therefore, when discussing a fault model, we consider the stuck-at fault model only.

In order to test a two-input AND or a two-input XOR gate against the stuck-at faults it is enough to apply three input test vectors (01, 10, 11).[1] We assume that the logic gates are atomic and the faults manifest at their I/O pins. It is an abstract fault model where a stuck-at fault does not mean the I/O pin is shorted to the power supply or ground. In fact that means when the line is applied with a certain logic value (0 or 1), it produces a logical error [12].

B. Modular Arithmetic over GF(2^m)

Each element of the field GF(2^m) can be represented as a polynomial of degree less than or equal to $m-1$, written as $A(x) = \sum_{i=0}^{m_w-1} A_i x^{iw} = (A_{m_w-1} \dots A_0)_{2^w}$ where A_i represents a single *digit* and $A_i \in [0, 2^w-1]$; m_w represents the number of digits and is equal to $\lceil m/w \rceil$ where w is a *digit-size*. A special case is when $w = 1$ and the representation of $A(x) = (A_{m-1} \dots A_0)_2$ is called a *bit* representation.

An addition (subtraction) of two elements in GF(2^m) is equivalent to a bit-wise XOR operation of these two elements and can be implemented very efficiently. A multiplication in GF(2^m) is multiplication modulo an irreducible polynomial that is used to define the field. Let $B(x)$ and $C(x)$ represent two elements in GF(2^m) and let $P(x)$ be an irreducible polynomial of degree m over GF(2) used to define the finite field. The multiplication of $B(x)$ and $C(x)$ over GF(2^m) is defined as

$$A(x) = B(x) \times C(x) \triangleq B(x) \cdot C(x) \bmod P(x) \ .$$

Algorithm 1 shows a digit-serial multiplication over GF(2^m). Due to the use of carry-free arithmetic, the algorithm is especially suitable for efficient hardware implementations. Step 3 of the algorithm performs a simple add and accumulate operation by adding a product $B_i C(x)$ to the shifted value of the accumulator $A(x)$. An eventual overflow is detected in step 4 and the following reduction is performed in step 5 by subtracting the multiple of modulus $P(x)$.

[1] An XOR gate can also be tested with any other three-element subset of (00, 01, 10, 11).

Algorithm 1 Digit-serial multiplication over GF(2^m).

Input: $B(x) = (B_{m_w-1} \dots B_0)_{2^w}$, $C(x) = (C_{m_w-1} \dots C_0)_{2^w}$ and $P(x) = x^m + p(x)$ where $p(x) = (P_{m_w-1} \dots P_0)_{2^w}$
Output: $A(x) = B(x)C(x) \bmod P(x)$.
1: $A(x) = 0$
2: **for** $i = m_w - 1$ downto 0 **do**
3: $\quad A(x) = A(x)x^w + B_i C(x)$
4: $\quad q_i = A(x) \operatorname{div} P(x)$;
5: $\quad A(x) = A(x) + q_i P(x)$
6: **end for**
7: return $A(x)$.

Note that for the case of $w = 1$ step 4 of the algorithm can be performed very efficiently by evaluating only the most significant bit of $A(x)$.

C. Random Pattern Testability

The random pattern testability has been extensively examined in the literature. Here, we outline basics and define some terms that will be used later in our discussion.

In order to estimate the number of (pseudo) random patterns required to test a CUT, we recall a well known testability procedure called Controllability Observability Procedure (COP) [7]. We define the *i*-controllability, and the observability of a node N. Then, by combining the two, we derive the fault detectability of the same node.

- The *i*-controllability for a node N, $C_i(\mathrm{N})$, $i \in \{0, 1\}$, is the probability of appearing the logic i on the node N.
- The observability of the node N, $O(\mathrm{N})$ is defined as the probability of observing the logic value of the node at one of the primary outputs.
- The fault detectability of the node N, $P_i(\mathrm{N})$, represents the probability of detecting stuck-at i fault at node N by a random pattern and is defined as $P_i(\mathrm{N}) = C_{1-i}(\mathrm{N})O(\mathrm{N})$.

Assume we have a circuit with k randomly inserted faults. Let the smallest fault detectability be called the critical fault detectability, denoted as $p = \min\{P_i \mid 1 \le i \le k\}$. The test quality q is defined as a probability that all faults are detected after the random test set of the length N is applied. Given the test quality q, Savir and Bardelin [15] derive a formula which provides an upper bound of the test length:

$$N \le \left\lceil \frac{\ln(e/k)}{\ln(1-p)} \right\rceil \qquad (1)$$

where $e = 1 - q$. In contrast to finding an exact value of N, this method requires less effort of analyzing the CUT prior to testing and thus has more practical importance [15].

III. The Proposed Testing Method

The testing method we propose enables configuration of digit serial multiplier over GF(2^m) as both TPG and SA, therefore providing the testing infrastructure for the other components. In addition, setting the multiplier to operate in the self-test mode it is possible to detect the faults of its own structure.

Our testing approach is based on generating pseudorandom patterns rather than providing a minimized, deterministic set of test vectors that is sufficient to perform the full testing against the stuck-at faults. By doing so, we introduce a minimal hardware overhead which makes this approach very suitable for the low-cost embedded devices. Furthermore, we reduce the effort of analyzing the CUT prior to testing in order to determine the minimized set of test vectors.

A. Building a Digit-Serial Multiplier

An efficient implementation of a digit-serial multiplier in finite fields of characteristic 2 has been an interesting research problem and there is a number of different designs proposed in the literature. What is of our main interest, are the multipliers used for the cryptographic applications and hence we mention some of the related work. Großschädel proposed a bit-serial multiplier performing multiplications in both types of field, GF(p) and GF(2^m) [6]. A similar hardware solution is also presented by Wolkerstorfer in [18]. A very compact Modular Arithmetic Logic Unit (MALU) that performs a digit-serial multiplication over GF(2^m) and a modular addition was introduced by Sakiyama [14]. The MALU is designed in a natural way to serve as an arithmetic core for the low-end devices. Therefore, we use the multiplier based on the MALU as a study case for our testing strategy.

The multiplier's basic cell consists of the full length AND and XOR arrays as shown in the right-hand part of Fig. 2a. The A1 array performs multiplication while the X1 (X2) array performs addition (reduction) modulo 2. The structure of the X2 array is determined by the irreducible polynomial which is of the form $P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1x + p_0$, where $p_j \in \{0, 1\}$, and the array is used for the reduction. Hence, the whole structure of the multiplier's basic cell is built out of the independent basic building blocks shown in Fig. 3.

A multiplier and a multiplicand are stored in the registers B and C respectively, and the register A acts as an accumulator. The register A is always shifted to the left by one bit (multiplication by 2), added to the partial product and then reduced by the polynomial $P(x)$ if needed. By using a single cell it is possible to perform a bit-serial multiplication only. In order to implement a digit-serial multiplier, w cells need to be connected in a serial manner as shown in Fig. 2a.

In this paper we explore a possibility to use the multiplier as a TPG and an SA for testing other modules of the cryptographic core. Furthermore, we analyze a self-testability of a digit-serial multiplier over GF(2^m) which is based on the original MALU. All the novel properties of the multiplier come at the very low hardware overhead and are shown in Fig. 2b. Section III-E further discusses the hardware overhead of the proposed design.

The proposed multiplier can perform the following modes of operation: Normal, Test Pattern Generator (TPG), Signature Analyzer (SA), and Self-Test (ST). Table I summarizes the functionality of the multiplier depending on the control signals CTRL0 and CTRL1 (when the control signal is low the multiplexer passes the upper bit; see Fig. 2b). Note that the signal CTRL0 remains the same during the Setup Phase and Configuration Phase, while the signal CTRL1 changes. Four modes of operation are further discussed in the following sections. We first consider a case when the multiplier works as a TPG (Section. III-B) and then we discuss about the multiplier in the SA and the ST mode (Sections. III-C and III-D, respectively).

Table I
Four Modes of Operation.

Configuration Mode	Setup Phase		Configuration Phase	
	CTRL0	CTRL1	CTRL0	CTRL1
Normal	0	0	0	0
TPG	1	1	1	0
SA	0	1	0	1
ST	1	1	1	0

B. Test Pattern Generator and Its Randomness

This section examines the randomness of the patterns produced by the proposed multiplier based TPG. While the randomness is an inherent property of cryptographic engines, the randomness of a single digit-serial multiplier is not so obvious. As a final test, we perform the NIST randomness testing composed of 15 statistical tests and show that our construction behaves very similar to an LFSR that is very often used as a pseudo-random test pattern generator.

In order to operate in the TPG mode, the multiplier needs to be properly initialized (Setup Phase) and modified (Configuration Phase) such that the least significant bit (LSB) of the shifting register B is updated from the most significant bit (MSB) of the register A. This tweak ensures the randomness of the register B even after m cycles are performed. The multiplexer M1, controlled by the signal CTRL0, is taking care of this modification. The multiplexer M2 only passes the MSB of the register B in the Configuration Phase (see Table I). Finally, the multiplexer M3 has no influence in this mode. However, in order to have only two control signals in total, the multiplexer M3 is controlled by CTRL0 and passes the A_{m-1} bit of the register A (this is further explained in Section III-D).

Our first aim is to evaluate the output probabilities of the proposed multiplier based TPG. To simplify the approach, we decompose the multiplier into segments of basic building blocks such as simple structures of AND and XOR gate arrays. As it is shown in Fig. 2a, the basic building block of a digit-serial multiplier over GF(2^m) can

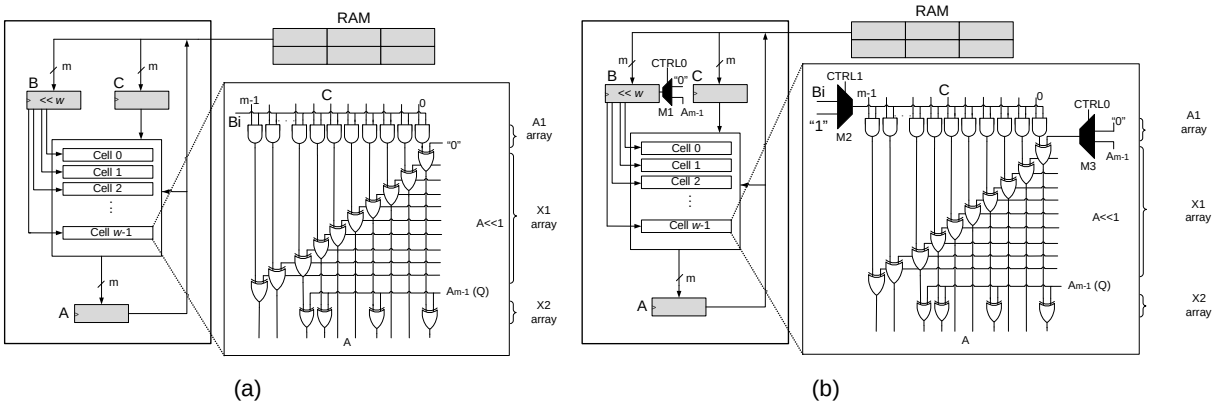

Figure 2. (a) Architecture of a digit-serial multiplier over GF(2^m). (b) Compact digit-serial multiplier architecture supporting the Normal, TPG, SA and ST mode.

be implemented as depicted in Fig. 3. It is a simple Reed-Muller structure [3] that contains one AND gate (A1) and two XOR gates (X1, X2).

With P_B, P_C, P_A and P_Q we denote the input probabilities of the circuit in Fig. 3. The probabilities of the intermediate signals X and Y are evaluated as $P_X = P_B P_C$ and $P_Y = P_A(1 - P_B P_C) + P_B P_C(1 - P_A)$, respectively.

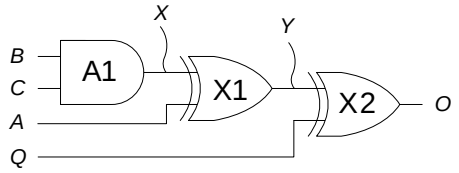

Figure 3. Basic building block of a digit-serial multiplier over GF(2^m).

Let us now compute a probability $P_O(i)$ of an arbitrary output bit of the multiplier being logic 1 in the i-th iteration (*i.e.* output probability). Since the registers A and C are updated from the same source (the multiplier's output), the probabilities $P_C(i+1)$, $P_A(i+1)$ and $P_Q(i+1)$ are equal to $P_O(i)$. Combining the equations for P_X and P_Y, we can finally compute the probability of an output bit being logic 1 at the end of the $(i+1)$-th iteration.

$$P_O(i+1) = 4P_B P_O(i)^3 - (4P_B+2)P_O(i)^2 + (2+P_B)P_O(i)$$

where P_B is a parameter and it holds

$$P_B = \begin{cases} \frac{1}{2} & \text{if } i \leq m; \\ P_B(i) & \text{if } i > m. \end{cases}$$

The previous assumption holds since, for $i \leq m$, B is a random number with a uniform distribution (during the Setup Phase we choose the values of B and C uniformly at random). After m cycles, the value of P_B is a function of i since the LSB of the register B is updated with the MSB of the register A. Let us now observe a general case where $P_B \in (0, 1)$ and analyze this iterative equation in order to see how the function of P_i behaves when the number of iterations goes to infinity.

Assuming that the function is convergent, $P_O(i+1)$ and $P_O(i)$ will converge to the same point as i goes to infinity. Therefore, we can write $\bar{P} = 4P_B\bar{P}^3 - (4P_B + 2)\bar{P}^2 + (2 + P_B)\bar{P}$, where $\bar{P}$ denotes the convergence point. A graphical representation of the function helps us to illustrate the estimation of the convergence point and is depicted in Fig. 4. It can be easily shown that the convergence point $\bar{P}$ is equal to $\frac{1}{2}$.

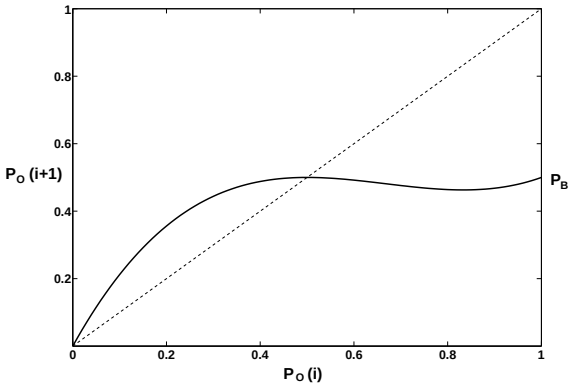

Figure 4. Graphical representation of the iterative function (2).

Let us first observe the case when $P_O(i) < \frac{1}{2}$. The function is clearly concave and always above the line $P_O(i+1) = P_O(i)$. This implies that $P_O(i+1) > P_O(i)$. Since Eq. (2) is iterative, the value of $P_O(i+1)$ will always be increasing, and hence it must hold $\lim_{i \to \infty} P_O(i+1) = P_O(i) = \frac{1}{2}$. Second, if $P_O(i) > \frac{1}{2}$, the function is always below the line $P_O(i+1) = P_O(i)$ and the value of $P_O(i+1)$ will always be decreasing in the following iterations. After a certain number of iterations, the value of $P_O(i+1)$ will become less than $\frac{1}{2}$ and will again converge to $\frac{1}{2}$ as i increases. Since if $P_O(i) = \frac{1}{2}$ then $P_O(i+1) = \frac{1}{2}$, this is indeed a convergence point. Finally, it is important to note here that for $P_O(i) = \frac{1}{2}$, the value of P_Y will be equal to $\frac{1}{2}$, independently of P_B.

We conclude that, no matter what the value of P_B is,

the output probability of an arbitrary output converges to $\frac{1}{2}$. As all the m output bits are computed independently, following the same principle, all the output probabilities converge to the same value. It turns out that, if the multiplier is initialized properly, the output probability converges to $\frac{1}{2}$ quite rapidly and ensures that after only a few iterations gets very close to a desired value. The only strict requirement is that $P_B = 1$ in the first iteration (Setup Phase). By satisfying this condition, the multiplier loads a random value of the register C as the initial seed. This is ensured by setting the CTRL1 signal to logic 1 during the Setup Phase (see Table I).

To examine the randomness of the output bits, we have implemented a multiplier over GF(2^{163}) and used the irreducible polynomial $P(x) = x^{163}+x^7+x^6+x^3+1$.[2] We used the NIST test battery composed of 15 statistical tests. As an input to the NIST test, we took the most significant bitstream of a size 10 Mb. The results are shown in Table II. For a comparison purpose, we have examined a randomness of a 163-bit LFSR with the same primitive polynomial and compared it to our proposed TPG. The global parameters are set such that if the result of the test is greater than 0.01 a bit sequence is considered as random.

Table II
NIST RANDOMNESS RESULTS OF THE MOST SIGNIFICANT BITSTREAM.

Test	TPG	LFSR
`Frequency`	0.534146	0.779188
`BlockFrequency`	0.739918	0.494392
`CumlativeSums`	0.911413	0.595549
`Runs`	0.350485	0.145326
`LongestRun`	0.911413	0.616305
`Rank`	0.739918	0.534140
`FFT`	0.350480	0.037560
`NonOverlappingTemplate`	0.911413	0.883171
`OverlappingTemplate`	0.350485	0.304126
`Universal`	0.213309	0.262240
`ApproximateEntropy`	0.350485	0.816537
`RandomExcursions`	0.819544	0.816537
`RandomExcursionsVariant`	0.788728	0.995711
`Serial`	0.534146	0.574903
`Linear`	0.000000	0.000000

Table II indicates the good randomness properties of the sequence for all the tests except the `LinearComplexity` test. In this test the input bitstream is split into N blocks of length M bits (*e.g.* $500 \leq M \leq 5000$). Now, by using Berlekamp-Massey algorithm [9] the test tries to find the shortest LFSR which can generate all the bits in the block i ($i = 1, \ldots, N$). An LFSR that is too short implies non-randomness. Since our TPG is a pseudorandom number generator, it is likely that there exist short LFSRs which can generate the M-bit blocks. This is generally true for most of the pseudorandom number generators. Furthermore, the same result is obtained in case of testing a 163-bit LFSR.

[2]This polynomial is standardized by the National Institute of Standards and Technology (NIST) and is especially suitable for the efficient implementation of ECC over GF(2^{163}) [1].

We conclude that the randomness properties of the bit stream produced by the multiplier in the TPG mode are at least as good as the LFSR bit stream that is very often used as a TPG in the BIST environments. Hence, the multiplier in the TPG mode can be used to produce the good pseudo-random test patterns. For more information about the NIST random tests we refer the reader to [1].

C. Multiplier as a Signature Analyzer

A signature analyzer is used to compress a sequence of the test responses into a single word and thus make the final comparison much more efficient. A good measure of the signature analyzer's quality is the probability of aliasing. It is a probability that the final signature corresponds to the expected one while some erroneous responses of a CUT have been masked and compacted into the signature analyzer [4].

By applying only a minor change, our proposed multiplier can be represented as an internal Multiple Input Shift Register (MISR) which is widely used as an SA in the BIST schemes. A transformation is performed by setting CTRL0 to logic 0 and CTRL1 to logic 1, and the multiplier acting as an SA is shown in Fig. 5. Again, the irreducible polynomial $P(x)$ is chosen as explained in the previous section and determines the X2 array. A response of the CUT is taken from the input C. If the bit errors of the input patterns are equally probable, the aliasing probability of such an SA tends towards $\frac{1}{2^r}$, where r is the length of the register [5]. In our case, the aliasing probability is equal to $\frac{1}{2^m}$.

When used in the SA mode, the multiplier is in the same state during the Setup Phase and the Configuration Phase. By setting the CTRL0 signal low, we ignore the influence of the XOR gate X1 at the LSB of A. In order to prevent masking of the erroneous responses we make the A1 array transparent to the CUT response C by setting the CTRL1 signal high (see Fig. 2b).

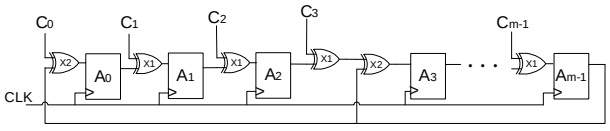

Figure 5. Digit-serial multiplier in the SA mode.

D. Self-Test

This section examines the use of the multiplier for its own testing (ST mode). The key idea is to configure the multiplier such that it concurrently acts as both a TPG and an SA. In the ST mode the outputs are fed back to the inputs providing the test patterns. Concurrently working as the SA, the multiplier compacts its outputs to the final signature.

In order to estimate the number of patterns required to test the multiplier, the pseudorandom testability properties of the structure from Fig. 3 are examined. Assuming that the primary inputs are chosen uniformly at random, which is justified in Section III-B, we can set the controllability of B, C, A and Q to $\frac{1}{2}$. This enables calculation of the fault detectability for all nodes in the circuit.

Table III
RANDOM TESTABILITY PROPERTIES OF THE BASIC BUILDING BLOCK.

Node	C_0	C_1	O	P_0	P_1
B	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$
C	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$
A	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
Q	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
X	$\frac{3}{4}$	$\frac{1}{4}$	1	$\frac{1}{4}$	$\frac{3}{4}$
Y	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
O	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$

Table III depicts random pattern testability properties of the circuit from Fig. 3. The critical fault detectability is $p = \min\{P_i\} = \frac{1}{4}$, which indicates that the circuit is highly testable with the pseudorandom test vectors. To estimate the test set length N for a given test quality of $q = 99\ \%$ and $p = \frac{1}{4}$ we recall Eq. (1) and obtain $N = 22$. In other words, in order to detect all possible 14 faults of the circuit from Fig. 3 with the probability of 99 %, we need to test the circuit with 22 random test vectors.

Since the multiplier is composed of the basic building blocks, we can reuse the testability analysis given above to estimate the number of patterns required to test the whole multiplier. The critical fault detectability of $\frac{1}{4}$ remains the same, but the number of critical faults increases. Additionally, by observing Fig. 2b we note that the input B of the basic building block is same for all m blocks. The same holds for the input Q. Therefore, to estimate the upper bound of the test length, we assume that the circuit has at most $k = 14m$ faults. We again set the test quality to $q = 99\ \%$ and $p = \frac{1}{4}$. For the case of $m = 163$, the upper bound of the test length is estimated to $N = 43$ random test vectors.

However, in ST mode the multiplier should operate in both TPG and SA mode. To ensure the correct response compaction, the input B should be fixed to logic one (see Section III-C). An eventual erroneous response will be masked whenever the B_i bit is low (see Fig. 2b). Assuming that the value of B is chosen uniformly at random, we expect approximately the same number of logic 1's and 0's, and therefore the number of test patterns needs to be doubled in order to ensure correct response compaction with the predicted aliasing probability. In our case, the number of necessary test vectors increases to 86.

Similar to the TPG mode, the multiplier in ST mode needs first to be initialized properly by setting both CTRL0 and CTRL1 high (Setup Phase). Later, in the Configuration Phase, the control signal CTRL0 is set to logic 1 and the CTRL1 signal is set to logic 0 (see Fig. 2b and Table I). By setting the CTRL0 signal high, the multiplexer M3 passes A_{m-1} bit of the register A and ensures testing of the rightmost XOR gate in the X1 array. Similarly, by setting the CTRL1 signal low, the multiplexer M2 passes the MSB of the register B and ensures the proper testing of the A1 array (see Fig. 2b).

To justify the theoretical approach we provide experimental results of the fault simulation and a fault coverage of the proposed multiplier scheme. A fault simulation is performed using the Synopsys Tetramax test suite [17]. The first fault simulation of the multiplier is performed by using an internal TPG provided by the simulation tool. The analysis in Section III has shown that 43 random input vectors are enough to fully test the multiplier with the confidence level of 99 %. We implement the multiplier over GF(2^{163}) using the UMC 0.13 μm CMOS standard cell library and apply 43 random test patterns in order to test it. The simulation is repeated 100,000 times and the fault coverage of 100 % occurs in 99,759 simulations (99.76 %). In this case, the fault coverage is defined as the percentage of detected faults out of all injected faults [17]. Therefore, the experimental results confirm the previous theoretical analysis.

The second simulation verifies the Self-Test approach described in Section III-D. The test patterns are now generated by the multiplier itself and fed back to it. Our analysis in Section III-D points out that in order to test the complete multiplier with the confidence level of 99 % and to ensure correct response compaction, we need to generate at most 86 random test vectors. However, the results show that 104 test vectors are necessary for the full fault coverage with the confidence level of 99 %. The explanation of this result lays in the fact that 86 random test patterns is a too short sequence for our TPG in order to meet the requirements in terms of randomness. On the other hand, the results prove that the multiplier is a complete self-testable design.

E. Hardware Overhead

This section presents the hardware overhead of the proposed test scheme. We have synthesized two digit-serial multipliers over GF(2^{163}) as proposed in Fig. 2a and Fig. 2b. The UMC 0.13 μm CMOS standard cell library and the Synopsys Design Vision tool version Y-2006.06 were used for the synthesis. The results are given in Table IV and show that by introducing an overhead of only 0.33 % (three dark multiplexers in Fig. 2b and some control logic) we can boost the functionality of the multiplier and have a TPG and an SA in the same circuit. Additionally, the multiplier becomes a self-testable design.

Table IV
HARDWARE OVERHEAD OF THE PROPOSED TEST SCHEME.

Design	Area (GE)	Overhead
Original Multiplier	4473	-
Proposed Multiplier	4488	0.33 %

Note that the hardware overhead represents only the additional hardware attached to the multiplier. Depending on the concrete application and the use of the multiplier, some more control logic to support the testing mode is also needed. Additionally, a comparator for comparing the signature obtained by the multiplier and the precalculated value is a necessary component. Since this heavily depends on the application we do not include it in our estimations.

IV. Conclusion

In this paper we proposed the low cost BIST based on digit serial multiplier over $GF(2^m)$. The multiplier is configured such that it acts as a Test Pattern Generator and a Signature Analyzer providing the BIST infrastructure for the surrounding components. Furthermore, it is shown that the multiplier is a self-testable design meaning that the errors in its own structure can be detected (ST mode). The introduced hardware overhead of only 0.33 % makes it highly suitable for low-end devices. The proposed TPG behaves very similar to an LFSR which is typically used for generating random patterns in the BIST environments. An ability to transform the multiplier into an MISR results in a very low aliasing probability and makes the multiplier a perfect choice for the SA.

The proposed scheme can be widely exploited in testing cryptographic applications, especially public-key crypto circuits. Those circuits process and store the sensitive data and hence, BIST is a highly suitable solution for fault detection. As the modular multiplication is a basic operation for most of the public-key algorithms, the modular multiplier is an integral part of these cryptographic cores, and therefore is a good choice for the BIST circuitry.

Having a multiplier over binary fields, limits the proposed approach to a certain number of public-key cryptosystems. Our future research will deal with the modular multipliers over integers and eventually will broaden the application to more of the existing cryptosystems.

Acknowledgment

This work was supported in part by K.U.Leuven-BOF (OT/06/40), by the European Commission under contract number ICT-2007- 216676 ECRYPT NoE phase II and FP7-238811 UNIQUE Project, by IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by FWO project G.0300.07.

References

[1] Digital Signature Standard (DSS) – FIPS 186-3. NIST Documents, 2009. Available at: http:\\csrc.nist.gov.

[2] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay. Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures. In *Progress in Cryptology - INDOCRYPT*, pages 226–238, 2008.

[3] D. E. Muller. Application of Boolean Algebra to Switching Circuit Design and to Error Detection. In *IRE Trans. Electron. Comput.*, volume EC-3, pages 6–12, 1954.

[4] M. Doulcier, M. L. Flottes, and B. Rouzeyre. AES-Based BIST: Self-Test, Test Pattern Generation and Signature Analysis. In *4th IEEE International Symposium on Electronic Design, Test and Applications*. IEEE, 2008.

[5] M.S. Elsaholy, S.I. Shaheen, and R.H. Seireg. A Unified Analytical Expression for Aliasing Error Probability Using Single-Input External and Internal XOR LFSR. In *IEEE Transactions on Computers*, pages 1414–1417, 2002.

[6] J. Grosschädl. A bit-serial Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^n). In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2162 in Lecture Notes in Computer Science, pages 206–223, Paris, France, May 13-16 2001.

[7] Wen-Ben Jone and S.R. Das. CACOP - A Random Pattern Testability Analyzer . In *Proceedings. The Sixth International Conference on VLSI Design*, pages 61–64, 1993.

[8] N. Koblitz. Elliptic Curve Cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[9] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[10] V. Miller. Uses of Elliptic Curves in Cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.

[11] G. Di Natale, M Doulcier, M. L. Flottes, and B. Rouzeyre. Low-Cost Self-Test of Crypto Devices. In *2nd Workshop on Dependable and Secure Nanocomputing*. IEEE/IFIP, 2008.

[12] Janak H. Patel. Stuck-At Fault: A Fault Model for the Next Millennium. In *ITC '98: Proceedings of the 1998 IEEE International Test Conference*, page 1166, Washington, DC, USA, 1998. IEEE Computer Society.

[13] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[14] K. Sakiyama. *Secure Design Methodology and Implementation for Embedded Public-key Cryptosystems*. PhD thesis, K.U Leuven, 2007.

[15] J. Savir and P.H. Bardell. On Random Pattern Test Length. *IEEE Transactions on Computers*, C-33(6):467 –474, june 1984.

[16] A. Schubert and W. Anheier. On Random Pattern Testability of Cryptographic VLSI Cores. In *Journal of Electronic testing: Theory and applications 16*. Kluwer Academic Publishers, 2000.

[17] Synopsys. *TetraMAX ATPG User Guide*, 2001.08 edition, 2001.

[18] J. Wolkerstorfer. Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$. In B.S. Kaliski Jr., Ç.Koç, and C. Paar, editors, *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2523 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[19] B. Yang, K. Wu, and R. Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. *International Test Conference*, pages 339–344, 2004.